# Setting up clusters of computing units to process several data streams efficiently

Daniel Millot, Christian Parrot

# Setting up clusters of computing units to process several data streams efficiently

Daniel Millot and Christian Parrot

Telecom sudParis - Institut Mines-Telecom - France
{Daniel.Millot, Christian.Parrot}@mines-telecom.fr

**Abstract.** Let us consider an upper bounded number of data streams to be processed by a Divisible Load application. The total workload is unknown and the available speeds for communicating and computing may be poorly a priori estimated. This paper presents a resource selection method that aims at maximizing the throughput of this processing. From a set of processing units linked by a network, this method consists in forming an optimal set of master-workers clusters. Results of simulations are presented to assess the efficiency of this method experimentally. Before focusing on the proposed resource selection method, the paper comes back on the adaptive scheduling method on which it relies.

**Keywords:** adaptive scheduling, parallel processing, master-worker model, load balancing, heterogeneous context, dynamic context

## 1 Introduction and related works

We consider an application that processes data acquired as streams, during an a priori unknown time-lapse. The throughput of the streams is supposed to be unlimited. Each data stream is arbitrarily associated to one processing unit in order to be processed. We assume that each data stream can be split into chunks as small as necessary for the scheduling and that each chunk can be processed independently of the others. Such applications, that can be found in many different domains [1, 2], look like a Divisible Load application [3], but we assume that the total workload is unknown.

Each processing unit which acquires a data stream can split the flow into chunks and distribute the chunks to be processed to other computation units, via a network. Each unit which receives a load can process it and send back the result of its processing; as a lot of data parallel applications [4, 5]. We assume that the communication speeds are high enough, in relation to the computation speeds, make the execution of this application to benefit from an execution in parallel.

According to the master-workers paradigm, the processing units which distribute the load act as masters and the computation units which do the processing act as workers [6]. Besides, we assume that the computation units have an unlimited buffering capability, unlike [7]. The resources, for both communicating and processing, can be heterogeneous as for [8, 9] and the durations of communicating and processing a chunk are supposed to be affine with respect to the chunk size.

Let us assume that the available communication speeds, available computation speeds and latencies that characterize these resources, and that we call execution parameters in the sequel, may be inaccurately specified; only estimates of the execution parameters are a priori available to the scheduler.

We assume that computation can overlap communication. We consider a 1-port bi-directional communication model between master and workers. This model allows a

communication from master to worker to overlap a communication from worker to master. But, with such a communication model, only one communication from, or to, one processing unit (master or worker) can be performed at the same time. This feature fits in with the behavior of message passing communications when messages are big enough [10].

We call "round" a sequence of consecutive actions leading the master to feed all the workers once and collect the corresponding results. As a result of the unawareness of the total workload, the minimization of the makespan is impossible and the scheduling must proceed in an iterative way, as the workload flows in. Therefore the scheduling has to be multi-round.

The processing of each data stream can be split into three phases. The start-up phase begins when the master starts to send the very first chunk to a worker and ends when each worker has received a chunk to process. Then begins the streaming phase which ends when the last data item in the stream has been acquired by the master. Eventually, the cleanup phase begins and it lasts until the master has received the very last result of processing o each worker.

When the cleanup phase begins, the total workload happens to be known and the makespan can be minimized by making the workers complete their work simultaneously [11]. In order to reduce as much as possible the cost of communication between workers to reallocate the chunks so as to balance their remaining load, when the cleanup phase begins, it is convenient to upper-bound the workload discrepancy between workers during the streaming phase. A usual way to do that is to make asymptotically periodic the distribution of the chunks, during the streaming phase. This strategy has the extra-advantage of facilitating the reduction of the risk of contentions when workers compete to communicate with the master [12].

Later on in this paper, we will consider that the start-up duration is negligible compared to the streaming phase duration; unlike [13, 14]. The legitimacy of this assumption increases with the ratio of the total workload to the load distributed at the very first round. Under this assumption it is reasonable to focus on the streaming phase and to aim at maximizing the throughput during this period.

In order to iteratively adjust the schedule to the possible inaccuracy of the specification of the execution parameter (and possibly even to their variation over time), it is relevant to arbitrarily set the frequency of the successive steps of this adaptation. The higher this frequency, in other words, the smaller the chunk sizes, the faster the adaptation; but unfortunately the longer the time wasted in latencies. We want the adjustment of the schedule for the next round to be based only on the measurement of durations like those of communicating or processing of the previous rounds. Indeed, this measurement is the best benchmark to estimate the actual value of the execution parameters.

Based on these remarks, the AS4DR (Adaptive Scheduling for Distributed Resources) method [15] automatically adapts the schedule to both: the heterogeneity of the workers and the poorness of the estimation of the execution parameters. But, it deals with only one data stream, processed by only one master-workers platform. This method sets the chunk size for each worker at each round by assuming that the execution parameters for the next round will be identical to the one of the previous round. Let $\alpha_{\mathrm{w,i}}$ be the size of the chunk sent to a worker w for round i. Let $\tau$ and $\sigma_{\mathrm{w,i}}$ be respectively the wanted and the measured time durations between the start of the sending of a chunk of size $\alpha_{\mathrm{w,i}}$ and the end of the reception of the corresponding result by the master. The basic idea of the

AS4DR multi-round method is to adapt $\alpha_{\mathrm{w,i}}$ according to:

$$\alpha_{\mathrm{w,i}} := \alpha_{\mathrm{w,i-1}} \frac{\tau}{\sigma_{\mathrm{w,i-1}}} \qquad \text{for i} > 1. \tag{1}$$

It has been proved [12] that with a multi-port communication model and linear costs for computations and communications, an asymptotic periodic schedule can be installed without knowing the execution parameters. This result can cope with an heterogeneous but steady-state context. Unfortunately this approach generates idleness of the workers between the processing of successive chunks, while the workers communicate with their master. Taking over this principle, AS4DR prevents this kind of idleness by making computation overlap communication. The asymptotic periodicity of the schedule with AS4DR remains when the costs for computations and communications are affine. With the aid of the estimate of the execution parameters and, before the launch of the AS4DR scheduler, a step called CIP (for Contentions and Idleness Prevention) determines a value for $\tau$, large enough to involve without contention all the workers into the scheduling [15]. It has been proved that AS4DR maximizes the CPU-efficiency of the workers in a heterogeneous but steady-state context [16]. Tests that have been performed experimentally assess these theoretical results, when considering a single master-workers platform. More, other tests have shown that AS4DR can adapt the schedule to not only the poorness of the estimates, but to the variation over time of the execution parameters as well [17]. But using an evermore increasing amount of resources makes the increase of the value of $\tau$ ultimately impossible, without reconsidering the hypothesis according to which the workload for a round is negligible with respect to the total workload. Section 2 reminds the way AS4DR schedules when considering only one data stream and only one master-workers platform.

To overcome this scaling-up difficulty, the constraint which bounds the number of data stream to one can be relaxed. Let M be an upper bound of the number of data streams. In this new context the preliminary step CIP is replaced by another one which, for a given value of $\tau$, selects workers to form master-workers clusters so that contentions are avoided. So, AS4DR is able to install an asymptotic periodic schedule for each data stream which, in the absence of contentions thus obtained, prevents the idleness of the workers; with the exception of latencies. The choice of the value of $\tau$ depends on several criteria. Of course, to reduce as much as possible the time wasted in latencies, the value of $\tau$ should be as large as possible. But, on the other hand, the smaller $\tau$, the faster the adaptation of the schedule to the poorness of the estimate of the execution parameters and also the more balanced the remaining workload between workers when the cleanup phase begins. When the execution parameters vary over time, the value of $\tau$ should be smaller than the smallest steady state period. When the upper bound M of the number of data streams is big enough with respect to the number of available workers, all workers can be involved into the processing. In this case, as all workers are fully used, the global throughput is maximum ; for a given value of $\tau$. On the contrary, if some workers are not selected (because of their lesser profitability for the global throughput), the throughput is not necessarily maximum, but the selected workers are fully used. Section 3 presents this preliminary step. Finally, we conclude in Section 4.

## 2 Presentation of the AS4DR method

To allow the overlapping between communication and computation, the AS4DR scheduler splits each chunk it has to deliver to a worker into two subchunks. So sending subchunks of arbitrarily chosen sizes $\dot{\alpha}_{\mathrm{w},1}$ and $\ddot{\alpha}_{\mathrm{w},1}$ to each worker w for the first round,

the AS4DR scheduler then sends to worker w, for each round i, two subchunks $\dot{\mathrm{s}}$ and $\ddot{\mathrm{s}}$ of respective sizes $\dot{\alpha}_{\mathrm{w,i}}$ and $\ddot{\alpha}_{\mathrm{w,i}}$, such that: $\dot{\alpha}_{\mathrm{w,i}} + \ddot{\alpha}_{\mathrm{w,i}} = \alpha_{\mathrm{w,i}}$.
Let $\theta_{\mathrm{w}}$ denotes the constant ratio between $\dot{\alpha}_{\mathrm{w,i}}$ and $\alpha_{\mathrm{w,i}}$. From estimates of the execution parameters, a preliminary step (for instance CIP) sets $\tau$ and $(\alpha_{\mathrm{w},1}, \theta_{\mathrm{w}})_{0 \leq \mathrm{w} \leq \mathrm{N}-1}$;
where N denotes the number of workers. The round i for worker w is composed of three phases: transmission in a row of the data from master to worker for subchunks $\dot{\mathrm{s}}$ and $\ddot{\mathrm{s}}$, worker computation on the received data, transmission in a row of the computation result from worker to master for subchunks $\dot{\mathrm{s}}$ and $\ddot{\mathrm{s}}$. It is worth noticing that the result corresponding to the $\ddot{\mathrm{s}}$ subchunk of some round is returned to the master just after the result corresponding to the $\dot{\mathrm{s}}$ subchunk of the next round has itself been returned. Let us denote $\dot{\mathrm{C}}_{\mathrm{w,i}}$ the measured time spent to process the subchunk $\dot{\mathrm{s}}$ of round i and $\mathrm{f_w}$ the latency of computation, for worker w. We define

$$\sigma_{\mathrm{w,i}} \equiv \frac{\dot{\mathrm{C}}_{\mathrm{w,i}} - \mathrm{f_w}}{\theta_{\mathrm{w}}} + 2\mathrm{f_w}. \tag{2}$$

When the communications between master and workers are contention-free, and if the workload is set with the aid of (1) and (2), then the effective duration of the rounds linearly converges to $\tau$ for any worker; whatever the initial (strictly positive) workload. So, the schedule established according to the AS4DR method is stable within the meaning of Lyapounov and, in addition, is asymptotically stable [16].

Besides, there exists [16] a lower bound and an upper bound: $\theta^{\min}_{\mathrm{w,i}}$ and $\theta^{\max}_{\mathrm{w,i}}$ such that, in the absence of contentions, AS4DR method prevents the idleness between the processing of two successive subchunks, if and only if, $\theta^{\max}_{\mathrm{w,i+1}} \geq \theta_{\mathrm{w}} \geq \theta^{\min}_{\mathrm{w,i+1}}$, $\forall$i.

To prevent contentions, the instant each worker accesses to the master is set far enough from the instants the others access too, by introducing time delays $\mathrm{d_w}$ before posting the very first subchunk to each worker w. Figure 1 illustrates the model of

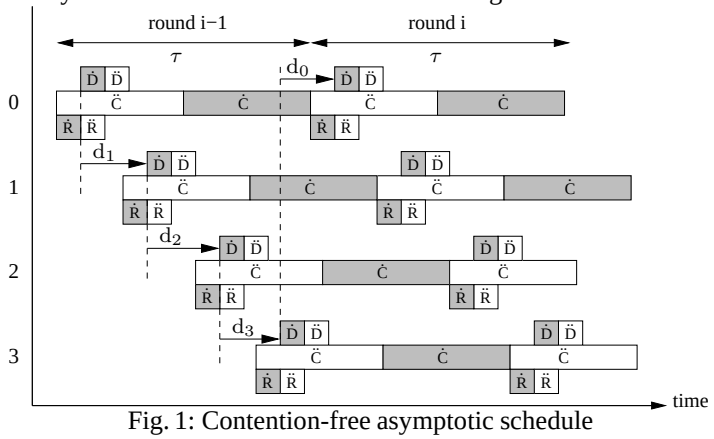

Fig. 1: Contention-free asymptotic schedule

asymptotic $\tau$-periodic (thus round-robin) schedule we are seeking, when N=4. Let us define the delay $\mathrm{d_w}$:

$$\mathrm{d_w} \equiv (1 + \lambda_{\mathrm{w}}) \max\left(\overline{\dot{\mathrm{D}}_{\mathrm{w}-1}} + \overline{\ddot{\mathrm{D}}_{\mathrm{w}-1}}, \overline{\ddot{\mathrm{R}}_{\mathrm{w}-1}} + \overline{\dot{\mathrm{R}}_{\mathrm{w}}}\right) \ (\mathrm{modulo\ N}); \tag{3}$$

where: $\lambda_{\mathrm{w}}$ stands for a positive constant factor (the greater the inaccuracy of the estimate of the execution parameters, the greater $\lambda_{\mathrm{w}}$), $\overline{\dot{\mathrm{D}}_{\mathrm{w}}}$ (resp. $\overline{\ddot{\mathrm{D}}_{\mathrm{w}}}$, $\overline{\dot{\mathrm{R}}_{\mathrm{w}}}$ and $\overline{\ddot{\mathrm{R}}_{\mathrm{w}}}$) denotes an a priori estimate of the time spent to communicate the $\dot{\mathrm{s}}$ data (resp. $\ddot{\mathrm{s}}$ data, $\dot{\mathrm{s}}$ result and $\ddot{\mathrm{s}}$

result), for worker w. These durations can be computed with the help of a priori estimates of the execution parameters [16].

Besides, the time intervals $\mathrm{d_w}$ should allow all the workers to be served during the first round, i.e. within a $\tau$ period. Thus, to prevent contention, it is sufficient that $\tau$ verify:

$$\sum_{\mathrm{w}=0}^{\mathrm{N}-1} \mathrm{d_w} \leq \tau. \tag{4}$$

# 3 Resource selection for AS4DR in a multiple data streams context

## 3.1 Method

Let $\mathcal{W}$ be the set of all available workers. Let $\mathrm{x_{m,w}}$ equals one if the worker w is selected in the cluster m, and $\mathrm{x_{m,w}}$ equals zero otherwise. As one worker can belong to one cluster at most,

$$\sum_{\mathrm{m}=0}^{\mathrm{M}-1} \mathrm{x_{m,w}} \leq 1, \quad \forall \mathrm{w} \in \mathcal{W}. \tag{5}$$

Let T be the global throughput,

$$\mathrm{T} = \sum_{\mathrm{m}=0}^{\mathrm{M}-1} \sum_{\mathrm{w}\in\mathcal{W}} \mathrm{t_w x_{m,w}}; \tag{6}$$

where $\mathrm{t_w}$, the potential throughput of worker w in the absence of idleness between the processing of any successive subchunks, verifies : $\mathrm{t_w} = \left(1 - 2\frac{\mathrm{f_w}}{\tau}\right)\mathrm{F_w}$;

where $\mathrm{F_w}$ and $\mathrm{f_w}$ respectively denote the available computation speed and the computation latency, of worker w. In order to prevent contentions, and thus to make the use of AS4DR possible, we have seen (4) that necessarily,

$$\sum_{\mathrm{w}\in\mathcal{W}} \mathrm{d_w x_{m,w}} \leq \tau_{\mathrm{m}}, \quad \forall\, 0 \leq \mathrm{m} \leq \mathrm{M}-1; \tag{7}$$

where $\tau_{\mathrm{m}}$ is the wanted duration for the rounds of cluster m. To form a set of clusters which maximizes the global throughput, when considering M data streams at most, we want to find $(\mathrm{x_{m,w}})_{0\leq \mathrm{m}\leq \mathrm{M}-1, \mathrm{w}\in\mathcal{W}}$ that maximizes T, given by (6), subject to constraints (7) and (5). This problem looks like a Multiple Knapsack problem. But as the value of $\tau_{\mathrm{m}}$ cannot be set a priori, since it depends on the workers selected for cluster m, the same wanted duration $\tau$ is set for the rounds of all clusters. This choice has the advantage of upper bounding the load discrepancy between the different clusters uniformly.

Besides, the time lag $\mathrm{d_w}$, defined by (3), depends on the worker w-1, the one that precedes w in the round robin distribution for the cluster which w belongs to. As this worker w-1 is a priori unknown, we need to reformulate the Multiple Knapsack problem to make the weight for worker w be independent from worker w-1. For that, let us tighten up the constraints of the problem. We define $\overset{\circ}{\mathrm{d}}_{\mathrm{w}}$, as follows:

$$\overset{\circ}{\mathrm{d}}_{\mathrm{w}} \equiv (1+\lambda_{\mathrm{w}}) \max\left(\overline{\mathrm{D_{max}}}, \overline{\ddot{\mathrm{R}}_{\mathrm{max}}} + \overline{\dot{\mathrm{R}}_{\mathrm{w}}}\right); \text{ where } \begin{cases} \overline{\mathrm{D_{max}}} \equiv \max\limits_{\mathrm{w}\in\mathcal{W}} \left(\overline{\dot{\mathrm{D}}_{\mathrm{w}}} + \overline{\ddot{\mathrm{D}}_{\mathrm{w}}}\right), \\ \overline{\ddot{\mathrm{R}}_{\mathrm{max}}} \equiv \max\limits_{\mathrm{w}\in\mathcal{W}} \overline{\ddot{\mathrm{R}}_{\mathrm{w}}}. \end{cases}$$

As $\mathrm{d_w}$ is smaller than $\overset{\circ}{\mathrm{d}}_{\mathrm{w}}$, the feasible space defined by the following constraints:

$$\sum_{\mathrm{w}\in\mathcal{W}} \overset{\circ}{\mathrm{d}}_{\mathrm{w}} \mathrm{x_{m,w}} \leq \tau, \quad \forall\, 0 \leq \mathrm{m} \leq \mathrm{M}-1, \tag{8}$$

is a subspace of the one previously defined by (7). So, the tightened problem we have to solve now is: find $(\mathrm{x_{m,w}})_{0\leq \mathrm{m}\leq \mathrm{M}-1, \mathrm{w}\in\mathcal{W}}$ that maximizes T, given by (6), subject to constraints (8) and (5).

Unfortunately, the coefficients $\mathrm{t_w}$, $\overset{\circ}{\mathrm{d}}_{\mathrm{w}}$ and $\tau$ are not positive integers, like for the classical Multiple Knapsack problem. To overcome this last complication, the resource

selection problem is reformulated. According to the machine number representation, $\mathrm{t_w}$, $\mathring{\mathrm{d}}_\mathrm{w}$ and $\tau$ are rational numbers. So, in order to make the problem be formulated like a Multiple Knapsack problem, (6) can be multiplied by the least common multiple of the denominators of $(\mathrm{t_w})_{w\in\mathcal{W}}$ whereas, (8) can be multiplied by the least common multiple of the denominators of $(\mathrm{d_w})_{w\in\mathcal{W}}$ and $\tau$. This new problem can be solved in a classical way by a Multiple Knapsack method. If the whole set of workers happens to be selected to participate in the processing, then the solution of this problem maximizes the global throughput; for a given wanted duration for the rounds $\tau$. Of course, as the presented method to select the workers is based on the underlying scheduling method: AS4DR, nothing allows one to claim that the global throughput is optimal if only a part of the whole set of workers happens to be selected. Of course, there exists a value of M big enough to make all the workers be ultimately involved into the schedule. Because of the successive transformations of the resource selection problem, the number of actually formed clusters is possibly no more minimal (but still smaller than M).

The next section is devoted to the experimental assessment of the resource selection method. The method described in [18] has been chosen to solve the Multiple Knapsack problem posed by the resource selection.

### 3.2 Experimental assessment

All the simulations, which results are presented in this section, have been conducted with the SimGrid framework [19]. We consider 10 sets $(\mathrm{S_k})_{0\leq \mathrm{k}\leq 9}$ of values for the execution parameters of the workers, given (speeds in bytes/second and latencies in seconds) in Table 1. Each of these sets is randomly a priori allocated to 100 workers among the 1000 available workers. Let us denote, $\mathrm{B_w^D}$ (resp. $\mathrm{B_w^R}$) the available communication speed of

| | computation | | communication master $\longrightarrow \mathrm{w_i}$ | | communication master $\longleftarrow \mathrm{w_i}$ | | number of |
|---|---|---|---|---|---|---|---|
| | speed | latency | speed | latency | speed | latency | workers |
| $\mathrm{S_0}$ | 1.0e+0 | 1.0e-2 | 1.0e+6 | 1.0e-2 | 1.0e+6 | 1.0e-2 | 100 |
| $\mathrm{S_1}$ | 1.0e+2 | 1.0e-1 | 1.0e+7 | 1.0e-3 | 1.0e+7 | 1.0e-3 | 100 |
| $\mathrm{S_2}$ | 1.0e+1 | 1.0e-3 | 1.0e+8 | 1.5e-2 | 1.0e+8 | 1.5e-2 | 100 |
| $\mathrm{S_3}$ | 1.0e+3 | 1.0e-2 | 1.0e+9 | 1.0e-2 | 1.0e+9 | 1.0e-2 | 100 |
| $\mathrm{S_4}$ | 1.0e+0 | 1.0e-3 | 1.0e+6 | 1.0e-2 | 1.0e+6 | 1.0e-2 | 100 |
| $\mathrm{S_5}$ | 1.0e+2 | 1.0e-1 | 1.0e+7 | 1.0e-3 | 1.0e+7 | 1.0e-3 | 100 |
| $\mathrm{S_6}$ | 1.0e+1 | 1.0e-4 | 1.0e+8 | 1.0e-2 | 1.0e+8 | 1.0e-2 | 100 |
| $\mathrm{S_7}$ | 1.0e+3 | 1.0e-2 | 1.0e+9 | 1.0e-2 | 1.0e+9 | 1.0e-2 | 100 |
| $\mathrm{S_8}$ | 1.0e+0 | 1.0e-3 | 1.0e+6 | 1.0e-3 | 1.0e+6 | 1.0e-3 | 100 |
| $\mathrm{S_9}$ | 1.0e+2 | 1.0e-2 | 1.0e+7 | 1.0e-2 | 1.0e+7 | 1.0e-2 | 100 |

Table 1: Reference values

| | |
|---|---|
| $\delta_0$ | 0.0 |
| $\delta_1$ | 0.1 |
| $\delta_2$ | 0.2 |
| $\delta_3$ | 0.3 |
| $\delta_4$ | 0.4 |
| $\delta_5$ | 0.5 |
| $\delta_6$ | 0.6 |
| $\delta_7$ | 0.7 |
| $\delta_8$ | 0.8 |
| $\delta_9$ | 0.9 |

Table 2: Dynamicity values

the link from the master to worker w (resp. from worker w to the master). To make $\mathrm{F_w}$, $\mathrm{B_w^D}$ and $\mathrm{B_w^R}$ varying over time, in a way that facilitates the correlation of the variations with their effects on the scheduling, each of these parameters can only take two values. According to the 10 profiles $(\mathrm{P_k})_{0\leq \mathrm{k}\leq 9}$ of variation shown in Figure 2, $\left(\mathrm{F_w},\ \mathrm{B_w^D},\ \mathrm{B_w^R}\right)$ alternatively takes the reference value (in Table 1): $\left(\left(\mathrm{F_w}\right)_{\mathrm{ref}},\ \left(\mathrm{B_w^D}\right)_{\mathrm{ref}},\ \left(\mathrm{B_w^R}\right)_{\mathrm{ref}}\right)$, and the perturbed value: $\left(\left(1-\delta_\mathrm{k}\right)\left(\mathrm{F_w}\right)_{\mathrm{ref}},\ \left(1-\delta_\mathrm{k}\right)\left(\mathrm{B_w^D}\right)_{\mathrm{ref}},\ \left(1-\delta_\mathrm{k}\right)\left(\mathrm{B_w^R}\right)_{\mathrm{ref}}\right)$; where $\delta_\mathrm{k}$ takes a strictly positive value given in Table 2. Each profile is randomly allocated to

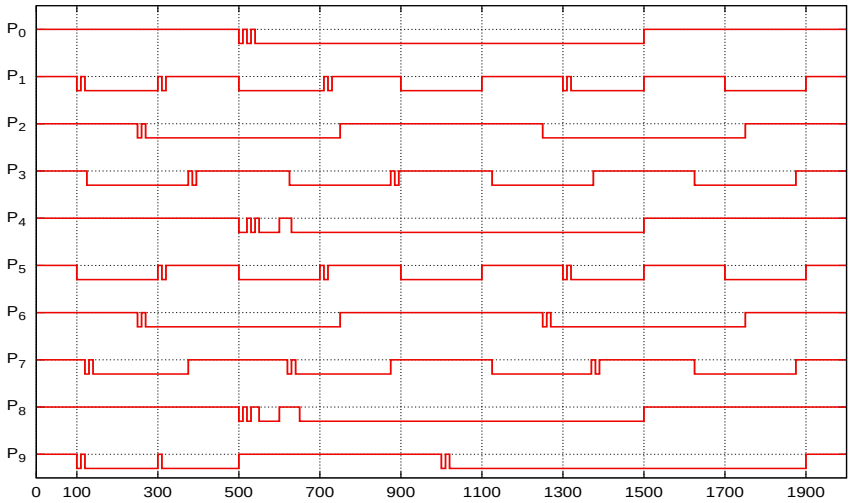

Fig. 2: Perturbed execution parameter as a function of time (in seconds)

100 workers. In this context, the coefficient $\delta_{\mathrm{k}}$ characterizes the variations of the execution parameters and is called "dynamicity" in the sequel; with $\delta_9$ the amplitude of the variation of the execution parameters is maximum, whereas it is minimum (steady state context) with $\delta_0$. For each simulation, the dynamicity is the same for all the workers.

Figure 3 shows the variation of the global throughput (in bytes/second) as a function of M, the upper bound of the number of data streams. For each value of M, the total number of selected workers is given. The execution parameters for this simulation are those of Table 1; dynamicity equals $\delta_0$, and the a priori estimate of the execution parameters equals their real values. As best workers are selected in priority, the greater the number of workers involved in the scheduling, the lower the increase of the throughput.

Figure 4 represents the mean value of the measured durations (in seconds) of the very first rounds: $\sigma$, for all selected workers, as a function of the elapsed time (in seconds), when $\tau$ equals 100 seconds, in steady state context (dynamicity=$\delta_0$). For this experiment, the initial workload is set to a value that could have resulted from a computation based on poor estimates of the execution parameters. This initial workload is set to -80% of the load computed with the aid of the real execution parameters; those of Table 1, when the latency for computation, for all the workers, is set to: 0.1, 0.5 and 0.9, successively. Figure 4 shows the convergence towards $\tau$ of the mean value of $\sigma$. As expected, the smaller the processing latencies, the faster the adaptation of $\sigma$ to $\tau$. Figure 4 illustrates the adaptation of the workload to the poorness of the estimation of the execution parameters.

Although the throughput remains the ultimate performance indicator, the throughput does not highlight the rate of time really spent in processing. Thus, let us define $\mathrm{CPU}_{\mathrm{eff}}$ the CPU-efficiency:

$$\mathrm{CPU}_{\mathrm{eff}} \equiv 1 - \frac{\text{CPU idleness}}{\text{elapsed time}}.$$

Figure 5 illustrates that the way to set the value of $\tau$ must depend on the computation latency; as long as these latencies are not negligible compared to $\tau$.This simulation has been performed in steady state context (dynamicity=$\delta_0$), with the full knowledge of the execution parameters of Table 1, but the computation latencies for all the workers are

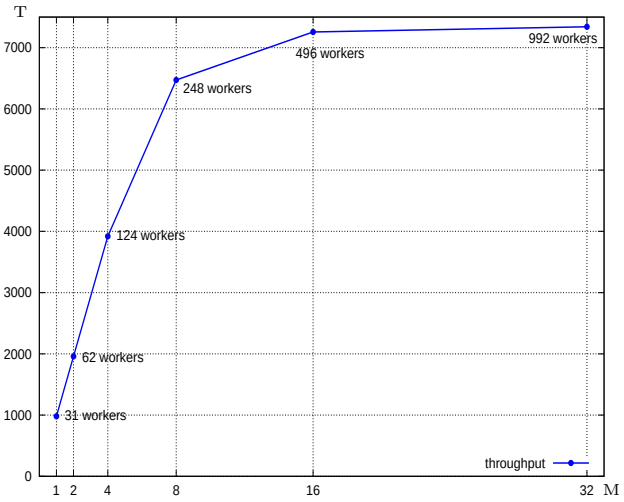


Fig. 3: Throughput as a function of the upper bound of the number of data streams: M

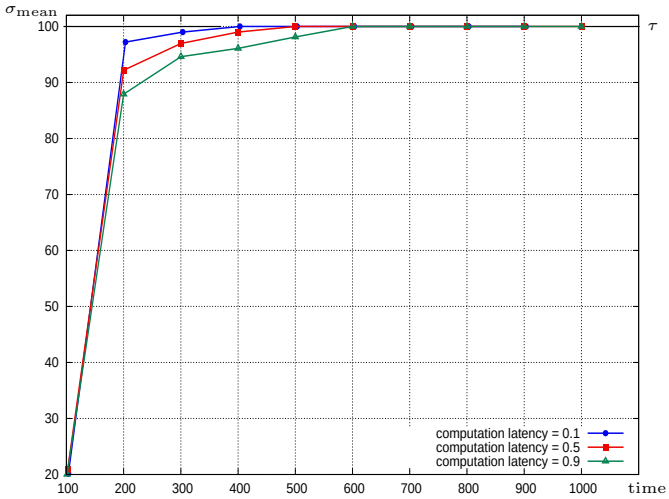


Fig. 4: Mean value of $\sigma$ as a function of elapsed time

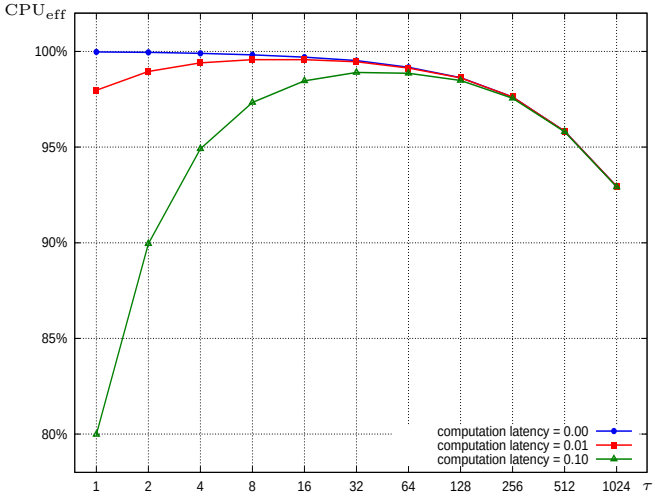


Fig. 5: $\mathrm{CPU_{eff}}$ as a function of $\tau$, when dynamicity = $\delta_0$

replaced by the values: 0.0, 0.01 and 0.1, successively. Figure 5 also reminds one that, according to the value of the computation latency, there exists an optimal value for $\tau$. The existence of such an optimal value is due to a compromise between two necessities. It is necessary to make $\tau$ as great as possible in order to minimize the time wasted in latencies; by reducing the number of rounds. On the other hand, the smaller the value of $\tau$, the earlier the worker starts to process.

In Figure 6 we can see that the higher the dynamicity, the smaller $\tau$ must be. Figure 6 also confirms that, even in a dynamic context, the smaller $\tau$ is, the higher the effects on the efficiency of the time wasted in latencies are. This simulation has been performed with several values of dynamicity, with the execution parameters of Table 1, but the CPU latencies for all the workers were replaced by the values: 0.0 and 0.01, successively.

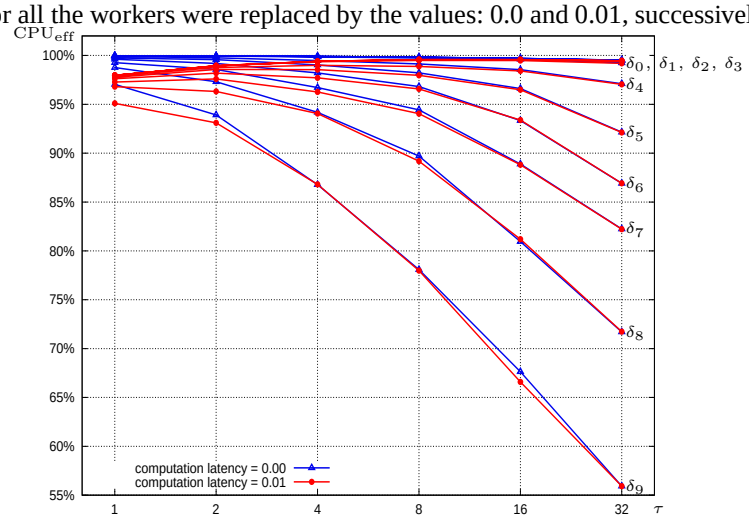


Fig. 6: $\mathrm{CPU}_{\mathrm{eff}}$ as a function of $\tau$, for several values of the dynamicity

## 4 Conclusion

This paper addresses the problem of setting up clusters of heterogeneous distributed resources to process several data streams of a Divisible Load application. As the total workload is unknown, a very first workload, corresponding to the wanted duration for the very first round: $\tau$, must be a priori set from estimates of the characteristics of the resources. To facilitate the contention prevention, the wanted duration for all the rounds and for all workers has been arbitrarily set to $\tau$. For the moment the value of $\tau$ is empirically set, according to both the poorness of the a priori estimate of the execution parameters and the frequency of their possible variation over time. If only a part of the whole set of available computation units are involved into the scheduling, then we can only claim that the selected computation units are fully used. If the whole set of available processing units are involved into the scheduling, then the presented method succeeds in maximizing the global throughput, for an a priori set value of $\tau$. Compared to using pure hardware performance figures, such as bandwidth or CPU frequency to adapt the workload at each round, the method has the extra advantage of taking into account characteristics of the software such as algorithmic complexity.

The way to adapt the workload at each round generates a risk of instability, when the execution parameters vary over time. A study of the stability of the method should lead to tighten up the way $\tau$ is set and should help to design new patterns of adaptation of the chunksize.

## References


1. C.Lee, M.Hamdi: Parallel image processing application in a network of workstation. Parallel Computing **21**(137-160) (1995)
2. D.Altilar, Y.Paker: An optimal scheduling algorithm for stream based parallel video processing. Proceedings of the 18th International Symposium on Computer and Information Sciences (ISCIS'03), Springer-Verlag (2003) 731–738
3. T.G.Robertazzi: Ten reasons to use divisible load theory. IEEE Computer **36(5)**(63-68) (2003)
4. D.Altilar, Y.Paker: An optimal scheduling algorithm for parallel video processing. Proceedings of the International Conference on Multimedia Computing and Systems, IEEE Computing Society Press (1998)
5. L.Dong, V.Bharadwaj, C.C.Ko: Efficient movie retrieval strategies for movie-on-demand multimedia services on distributed networks. Multimedia Tools and Applications **20**(2) (2003) 99–133
6. O.Beaumont, H.Casanova, A.Legrand, Y.Robert, Y.Yang: Scheduling divisible loads on star and tree networks: results and open problems. IEEE Transactions on Parallel and Distributed Systems **16**(3) (Mar. 2005) 207–218
7. M.Drozdowski, P.Wolniewicz: Optimizing divisible load scheduling on heterogeneous stars with limited memory. European Journal of Operational Research **172**(2) (2006) 545–559
8. A.L.Rosenberg, R.C.Chiang: Toward understanding heterogeneity in computing. Proceeding of the 24th International Parallel and Distributed Processing Symposium (IPDPS'10). Volume 1., IEEE Computing Society Press (April 2010) 1–10
9. O.Beaumont, L.Marchal, Y.Robert: Scheduling divisible loads with return messages on heterogeneous master-worker platforms. Lecture Notes in Computer Science **3769** (2005) 498–507
10. T.Saif, M.Parashar: Understanding the behavior and performance of non-blocking communications in mpi. Proceedings of the 9th international Euro-Par Conference (Euro-Par 2004), Springer-Verlag (2004) 173–182
11. V.Bharadwaj, D.Ghose, V.Mani, Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computing Society Press (1996)
12. M.Drozdowski: Selected problems of scheduling tasks in multiprocessor computing systems. PhD thesis, Instytut Informatyki Politechnika Poznanska, Poznan (1997)
13. V.Bharadwaj, D.Ghose, V.Mani: Multi-installment load distribution in tree networks with delays. IEEE Transactions on Aerospace and Electronic Systems **31**(2) (1995) 555–567
14. Y.Yang, H.Casanova: Extensions to the multi-installment algorithm: Affine costs and output data transfers. Technical Report Tech. Rep. CS2003-0754, Dept. of Computer Science and Engineering, University of California, San Diego (July 2003)
15. D.Millot, C.Parrot: Scheduling on unspecified heterogeneous distributed resources. Proceeding of the 25th International Symposium on Parallel and Distributed Processing Workshops (IPDPSW'11). Volume 1., IEEE Computing Society Press (May 2011) 45–56
16. D.Millot, C.Parrot: Fundamental results on the AS4DR scheduler. Technical Report RR-11005-INF, TELECOM sudParis, Évry(France) (December 2011)
17. D.Millot, C.Parrot: Some tests of adaptivity for the AS4DR scheduler. Proceedings of the 41th International Conference on Parallel Processing (ICPP'12), IEEE Computing Society Press (September 2012) 323–331
18. D.Pisinger: An exact algorithm for large multiple knapsack problems. European Journal of Operational Research **114**(3) (1999) 528 – 541
19. H.Casanova, A.Legrand, M.Quinson: Simgrid: a generic framework for large-scale distributed experiments. Proceedings of the 10th International Conference on Computer Modeling and Simulation (ICCMS'10), IEEE Computing Society Press (March 2008) 126–131